Unbounded Hardware Transactional Memory for a Hybrid DRAM/NVM Memory System

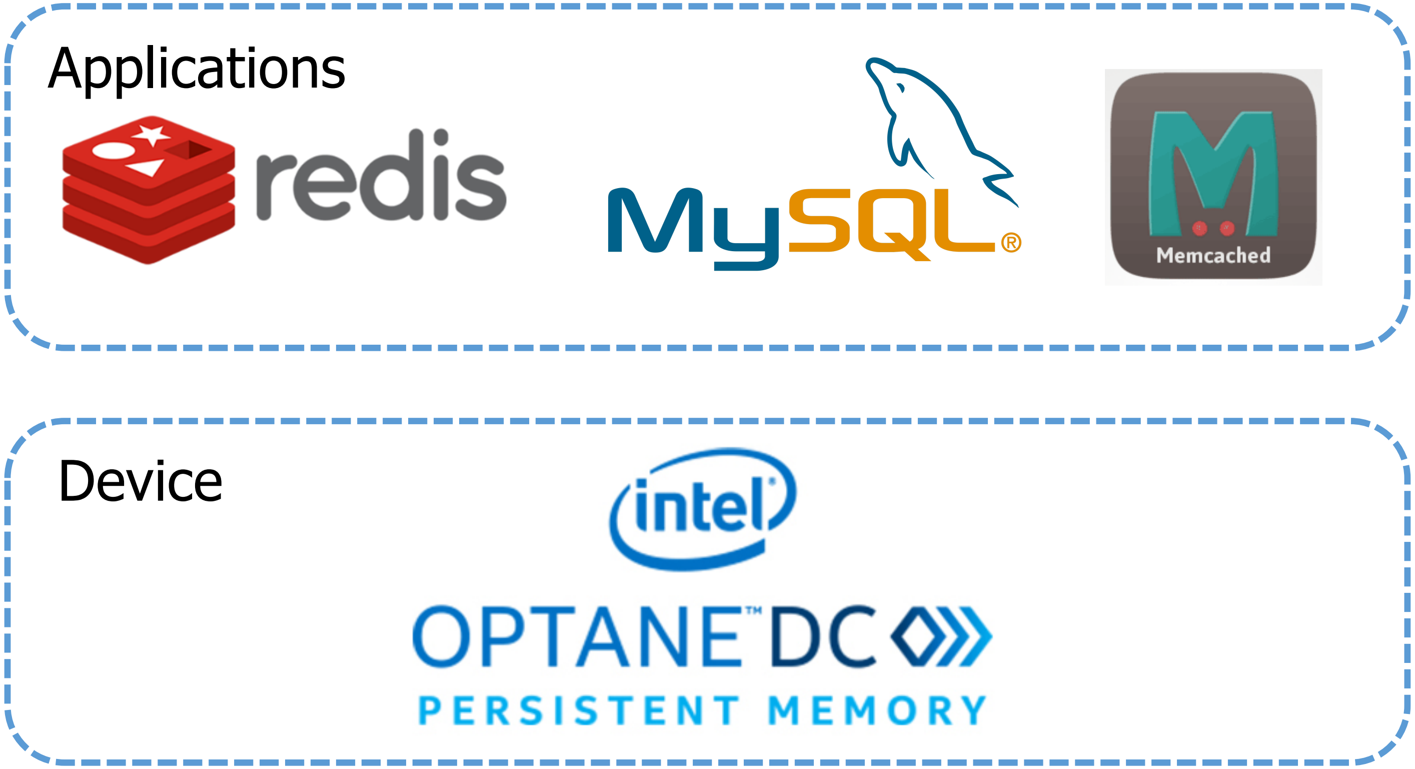
Jungi Jeong[§], Jaewan Hong[†], Seungryoul Maeng[†],
Changhee Jung[§], and Youngjin Kwon[†]

[§] Purdue University
[†] KAIST

International Symposium on Microarchitecture (MICRO) 2020

Persistent Memory (PM)

- Larger capacity than DRAM
- Non-Volatile
- Direct-Access via CPU load/store instructions

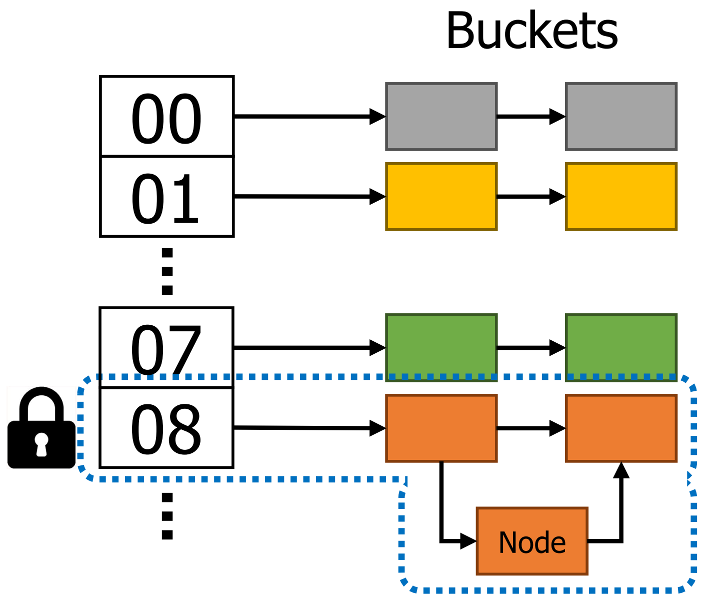


Applications

Device

Persistent memory as persistent memory

- PM programming example: persistent hash table[1,2]



[T0]: Add new node 🟧

```
Prev->Ptr = Node;
Node->Ptr = Next;
```

Atomicity: all or nothing
Durability: written to NVM

[T1]: Add new node 🟧

```
Prev->Ptr = Node;
Node->Ptr = Next;
```

Consistency: App-Dependent

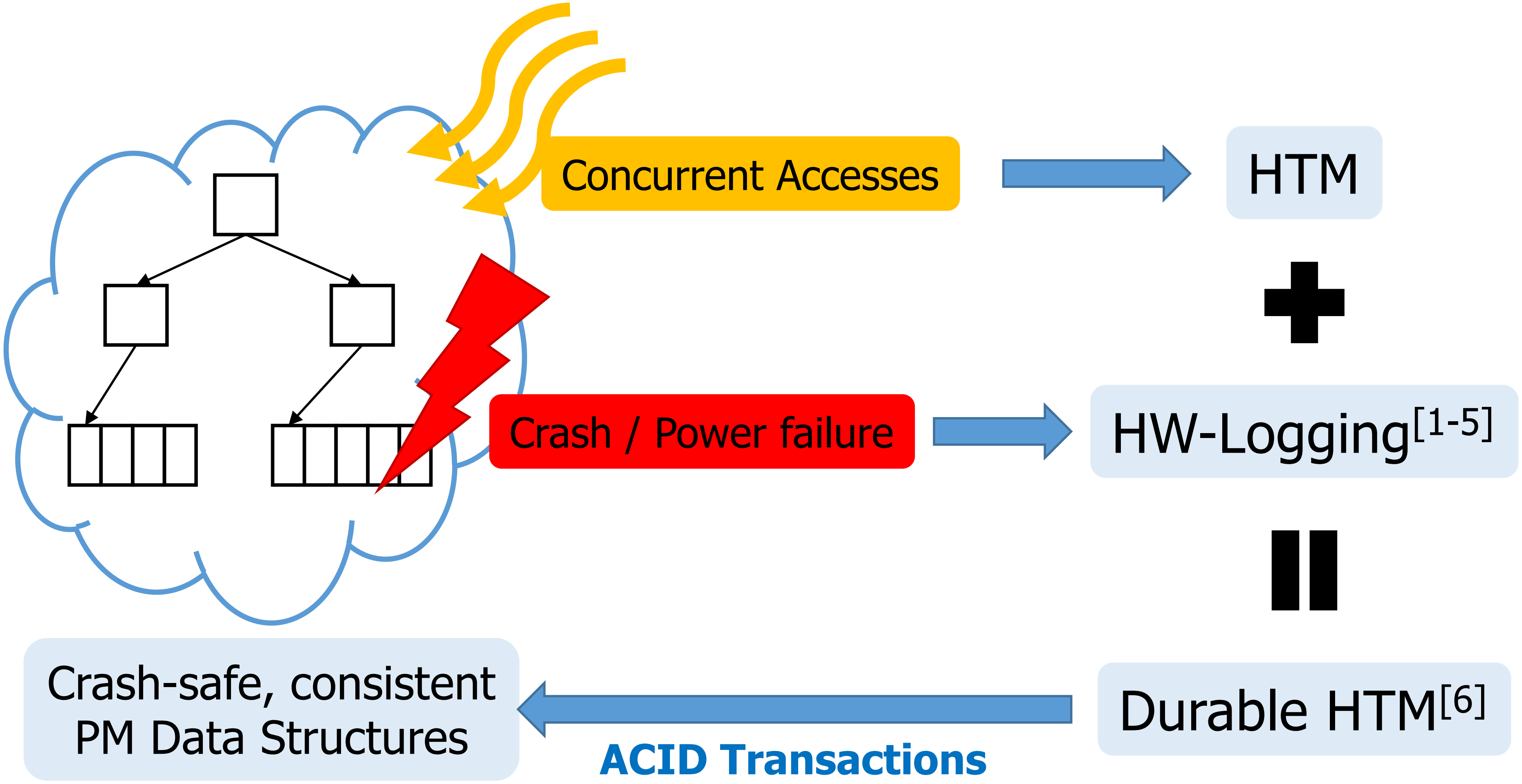
Isolation: serialize if needed

PM Programming requires ACID
→ **Durable Transactions**

[1] Zuo et al., OSDI 2018.

[2] Nam et al., FAST 2019.

HW support for PM programming



Concurrent Accesses

HTM

Crash / Power failure

HW-Logging[1-5]

Durable HTM[6]

Crash-safe, consistent PM Data Structures

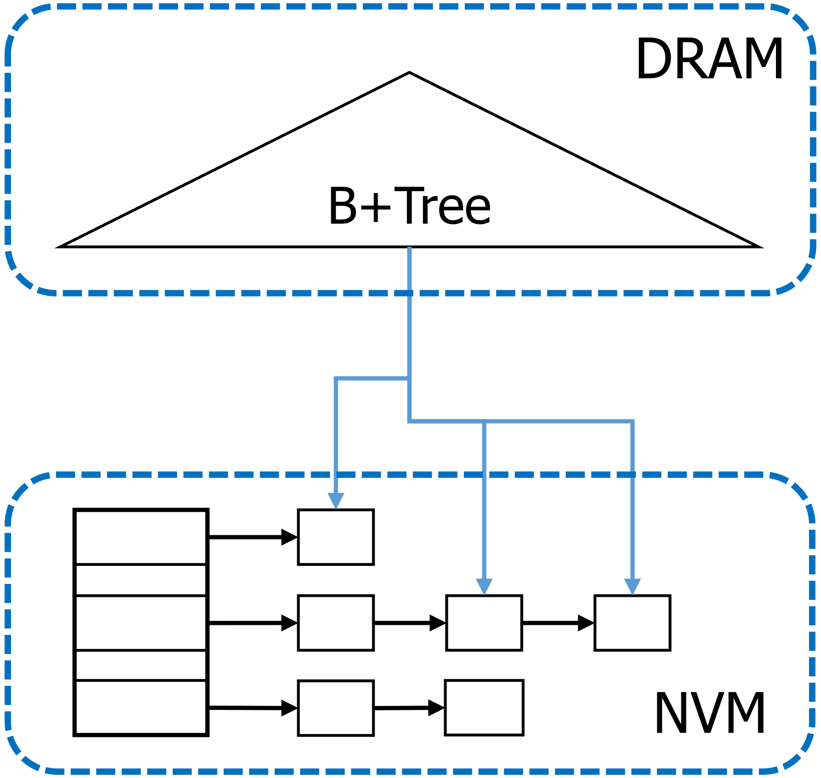
ACID Transactions

[1] Doshi et al., 2016.
[2] Joshi et al., 2017.
[3] Shin et al., 2017.
[4] Ogleari et al., 2018.
[5] Jeong et al., 2018.
[6] Joshi et al., 2018.

Limitations of Previous Work

1. NVM data only in durable transactions

- Recent applications strive to use both DRAM/NVM
  - Ex) Index in DRAM, data in NVM[1,2]



→ To accelerate search

+ To simplify programming

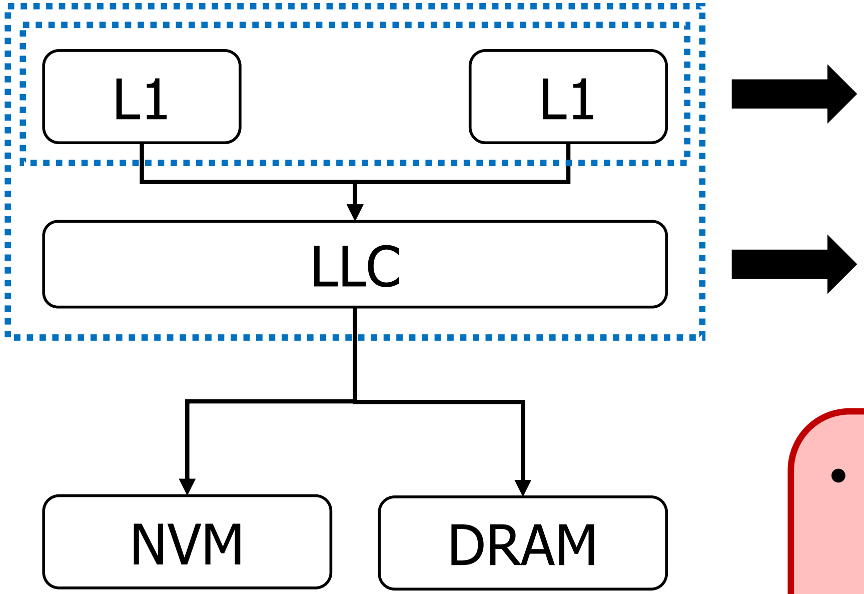
= **Hybrid DRAM/NVM Support**

[1] Yang et al., 2015.
[2] Xia et al., 2017.

Limitations of Previous Work

2. Limited transaction boundary

- Previous works are bounded or inefficient



Commercial HTMs (e.g., Intel):
L1-bounded

State-of-the-art[1]:
LLC-bounded

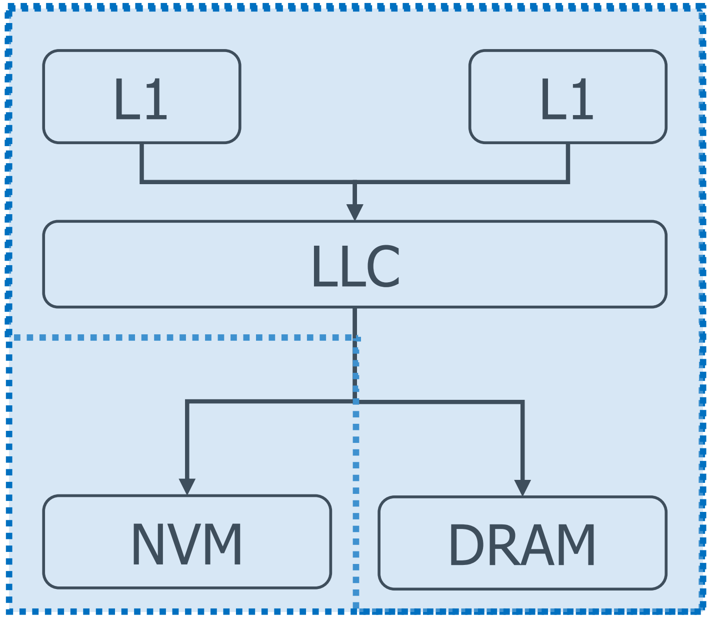
- Fallback path
  : ↑ **programming complexity**
- Serialization
  : ↓ **transaction concurrency**

[1] Joshi et al., 2018.

Limitations of Previous Work

2. Limited transaction boundary

- Previous works are bounded or inefficient



Commercial HTMs (e.g., Intel):
L1-bounded

State-of-the-art[1]:
LLC-bounded

DRAM-only unbounded HTM[2,3]:
Severe **false-positives**

Unlimited Conflict Detection

: at all memory hierarchies
: with low false-positive rates

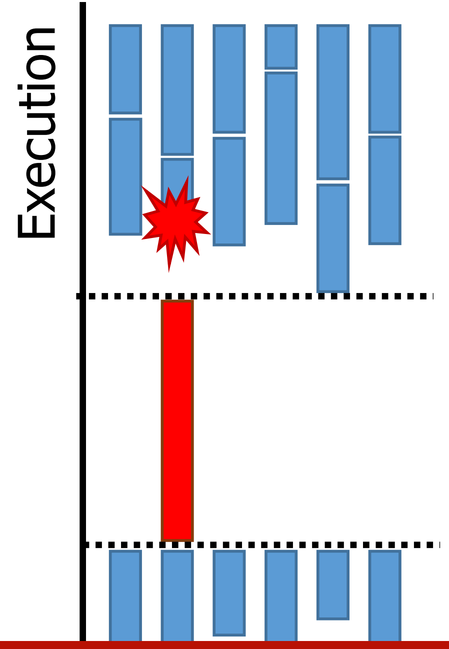
[1] Joshi et al., 2018.
[2] L. Yen et al., 2007.
[3] A. Shriraman et al., 2008.

Why Overflows Matter?

- Overflows are unpredictable and unexpected
  - Small transactions can be overflowed[1-4]
  - Overflowed transaction tends to repeatedly overflow if retried
- Current solution: user-defined handler[5]

```
while (1) {
  int status = tx_begin();
  if (status == XBEGIN_STARTED) {
      /* transaction body:
         fast-path */
      tx_end(); return;
  }
  if (!(status & XABORT_RETRY)) {
      /* user-defined handler
         for overflow:
         slow-path */
  }
  /* User-defined handler
```

User-defined handler
→ programmer burden



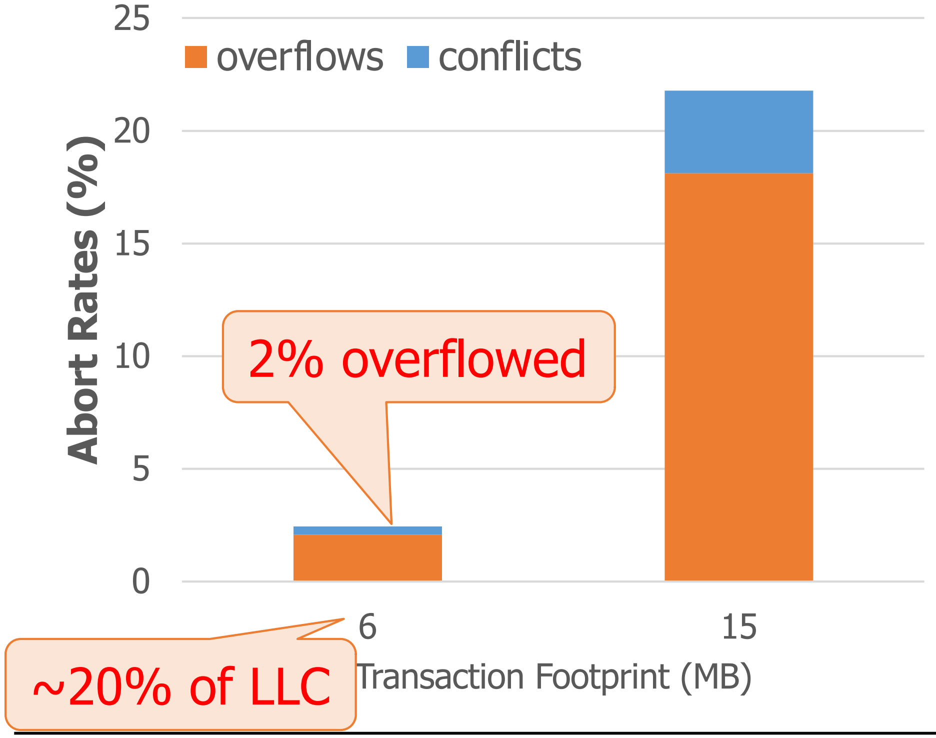
Serialized execution
→ throughput degradation

[1] A. Wang et al., PACT 2012.
[2] C. Jacobi et al., MICRO 2012.
[3] T. Karnagel et al. HPCA 2014.
[4] A. Joshi et al. ISCA 2018.
[5] Intel TSX Software Development Guide.

How Severe Capacity Overflows?

- B-tree benchmark / 16-core and 32MB LLC (2MB per core)
- Two value sizes: small (1KB, 50%) and large (64KB, 50%)

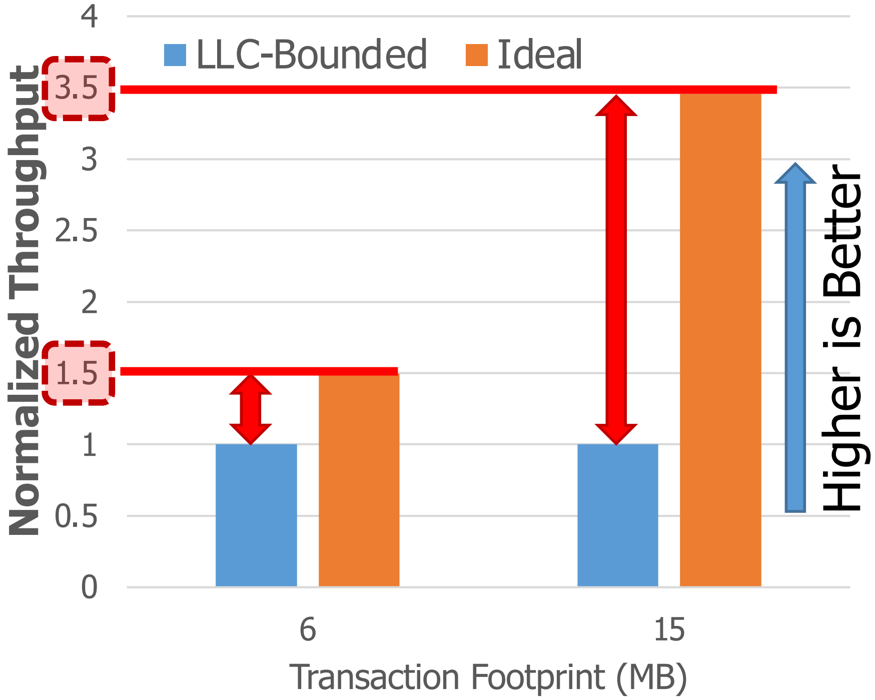
Overflows happen with much smaller footprints



2% overflowed

~20% of LLC

Overflows severely degrade application throughput



Overview

Durable HTM for convenient PM programming

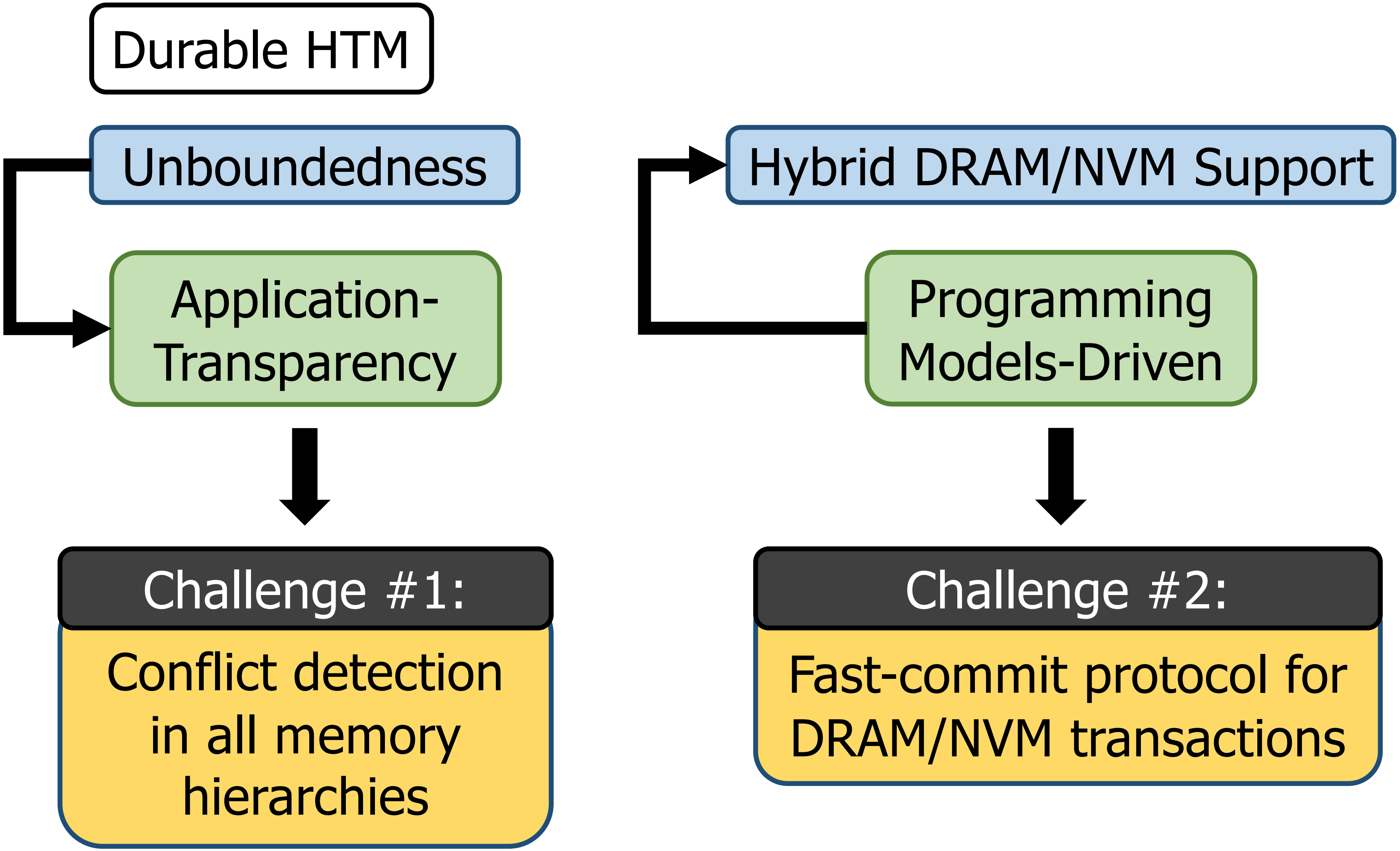
+ Unboundedness + Hybrid DRAM/NVM Support

=

UHTM:

Unbounded HTM for Hybrid DRAM/NVM System for **efficient, simpler** PM programming

Overview


Durable HTM

Unboundedness

Application-Transparency

Challenge #1:

Conflict detection in all memory hierarchies

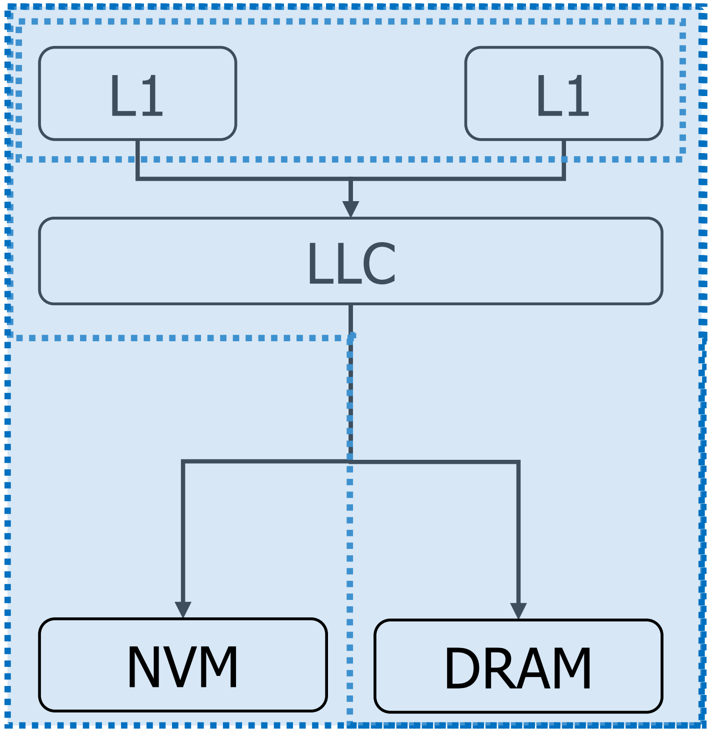
Hybrid DRAM/NVM Support

Programming Models-Driven

Challenge #2:

Fast-commit protocol for DRAM/NVM transactions

Unlimited Conflict Detection



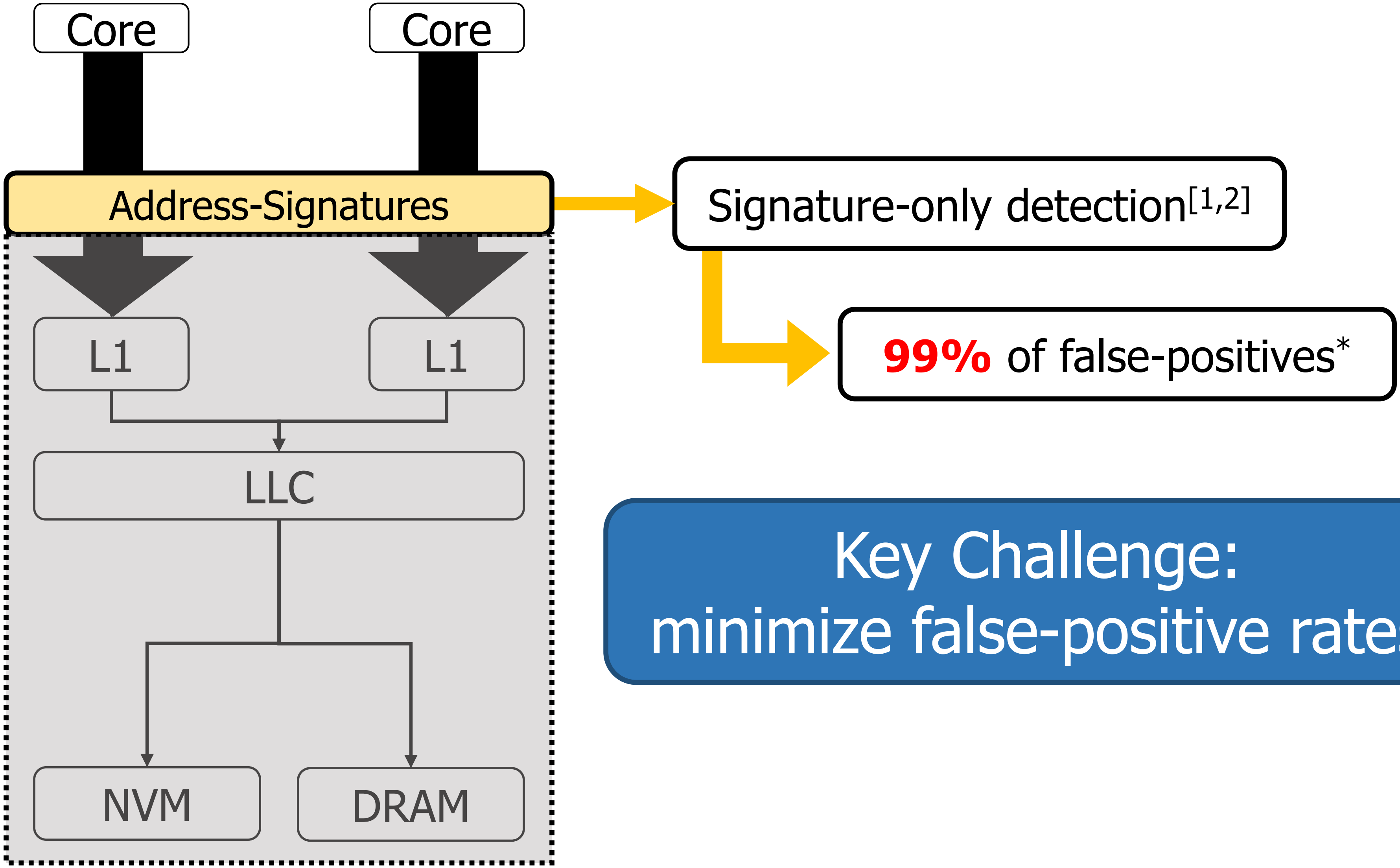
Commercial HTMs (e.g., Intel): **L1-bounded**

State-of-the-art[1]: **LLC-bounded**

DRAM-only unbounded HTM[2,3]

[1] Joshi et al., 2018.
[2] L. Yen et al., 2007.
[3] A. Shriraman et al., 2008.

Unlimited Conflict Detection



Signature-only detection[1,2]

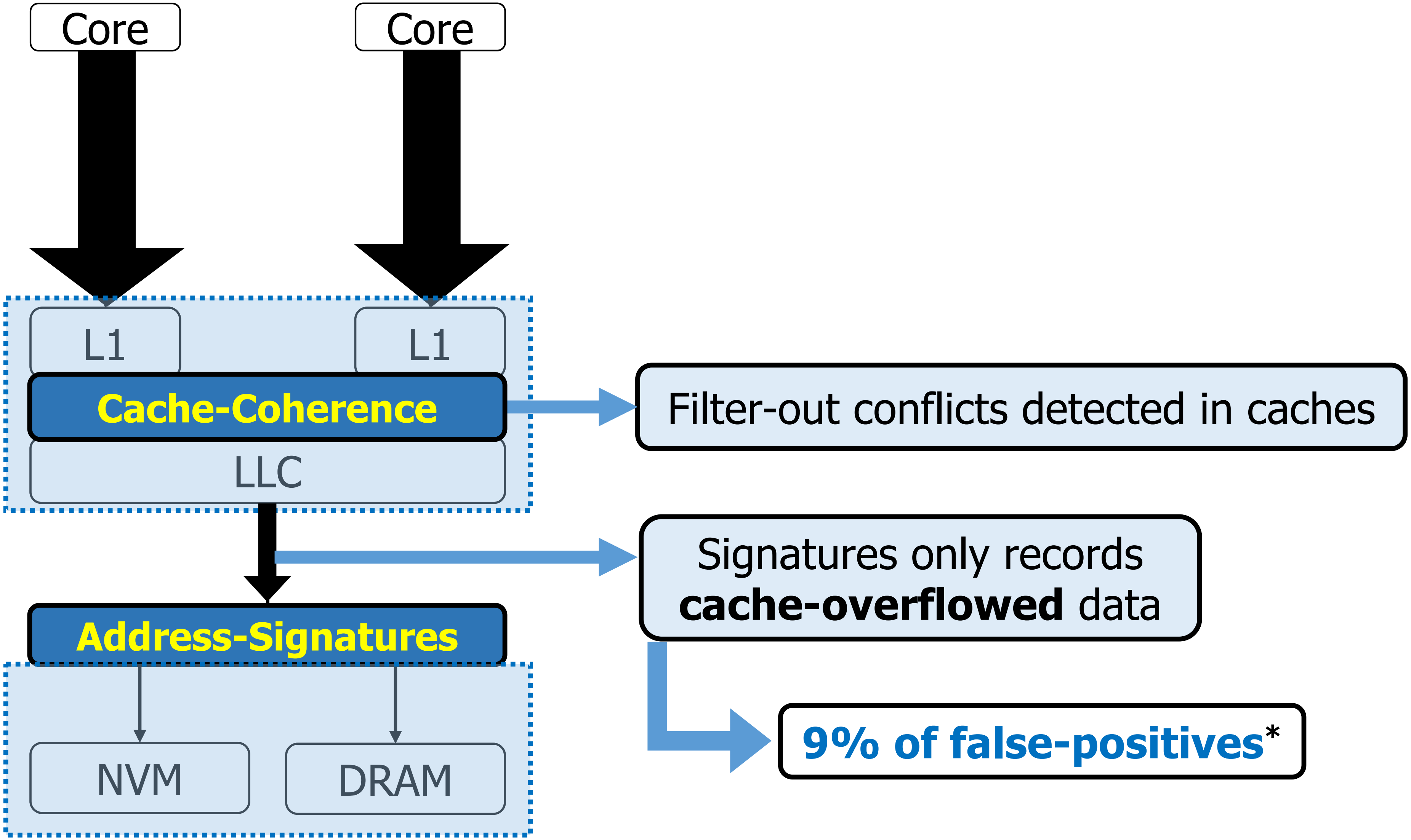
99% of false-positives*

Key Challenge:
minimize false-positive rates

[1] L. Yen et al., 2007.
[2] A. Shriraman et al., 2008.

* Use 512-bit signatures.

Unlimited Conflict Detection



Filter-out conflicts detected in caches

Signatures only records **cache-overflowed** data

9% of false-positives*

[*] Use 512-bit signatures.

Can we reduce false-positives further?

- Large signature (16k-bit) reduces to 9% (from 26%)
- But, too much space overheads
  - E.g., 16-core requires 64KB for signatures

Observations

- #1: conflicts by non-Tx & background processes
  - No possible conflicts, but false-positives produce false-conflicts
- #2: % false-conflicts ~= O(# signatures) ~= O(# cores)
  - Higher possibility to produce false-positives in signatures

Signature Isolation

- Conflict domain
  - A group of transactions that potentially have conflicts each other
  - Assign transaction-group ID to each thread
  - Ex) multiple threads in a process

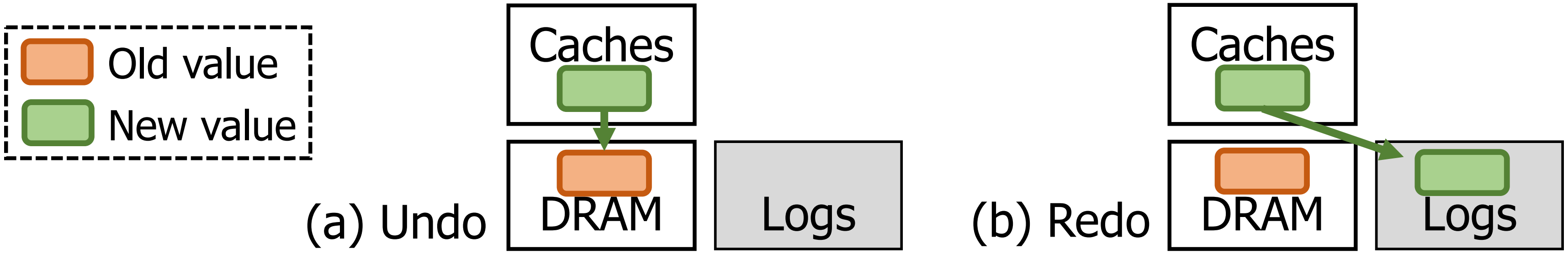
- Signature checking happens within the conflict domain
  - With the same group
  - Non-Tx & background ones not participating in conflicts checking

UHTM

Hybrid DRAM/NVM Support

- DRAM only requires atomicity
  - No cache-flush
  - Logging only **overflowed** blocks (HTM handles within caches)

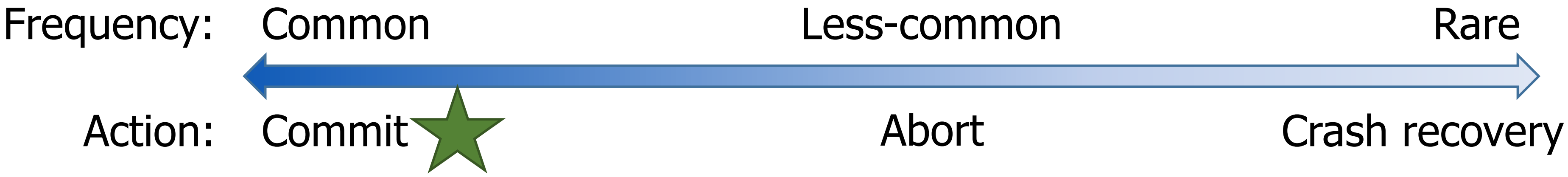
- Undo vs. Redo when LLC-overflow of DRAM data:



Load miss on LLC:	Direct reads	Search logs first
When commit:	Commit immediately	Copy before commit
When abort:	Copy before abort	Abort immediately

UHTM

Hybrid DRAM/NVM Support



Frequency: Common Less-common Rare

Action: Commit Abort Crash recovery

- Hybrid eager/lazy-approach for FAST-commit
  - DRAM data ➔ Undo-logs (eager)
  - NVM data ➔ Redo-logs* (lazy)

Placing cache-flush to NVM out of the critical path

- Crash recovery guaranteed by NVM redo-logging

* Jeong et al., MICRO 2018.

Methodology

- Gem5 simulator

Processor	16-core, In-Order, 2GHz, x86
L1 I/D cache	Private, 32KB, 8-way
L2 cache	Shared, 32MB, 16-way
DRAM	Read/Write: 82ns
NVM	Read: 175ns, write: 94ns

- Benchmarks

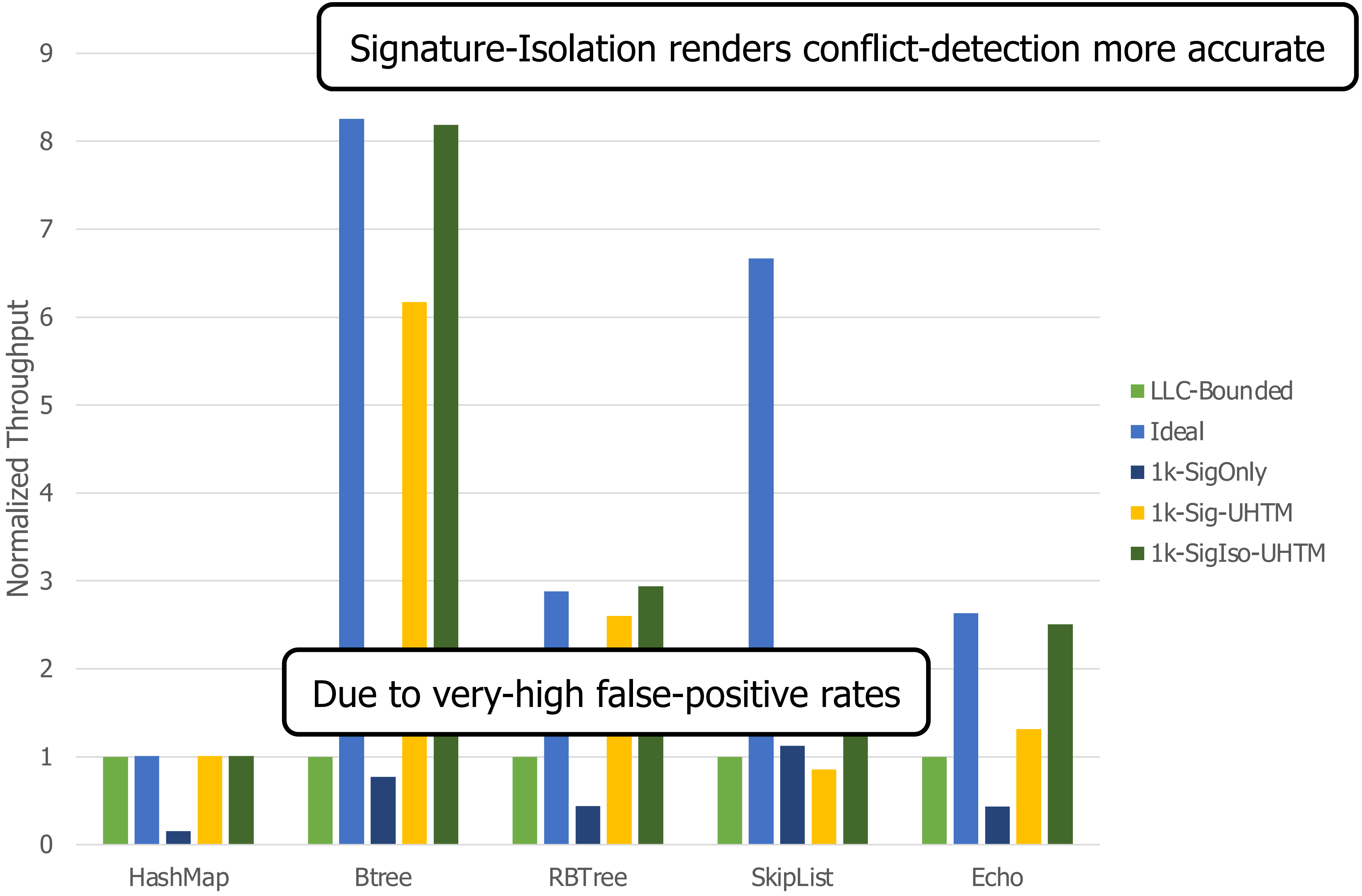
PMDK	HashMap, B-Tree, RB-Tree, SkipList
KV-Store	Hybrid-Index[1], Echo

* 4 processes with 4 threads

- Comparing schemes
  - LLC-Bounded[2]: Abort and restart if Tx overflows from LLC
  - SigOnly-HTM: Signature-only unbounded HTM
  - Ideal: Ideal unbounded HTM with no false-positives
  - UHTM

[1] F. Xia et al., 2017.
[2] A. Joshi et al., 2018.

Evaluation

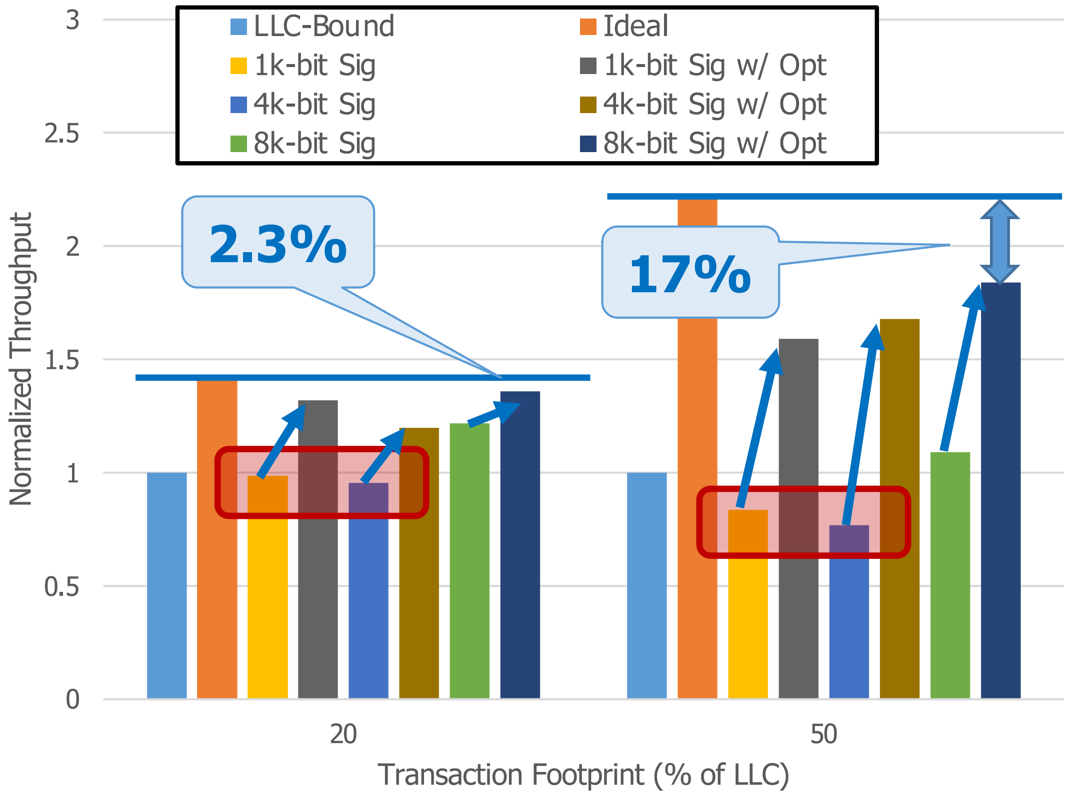


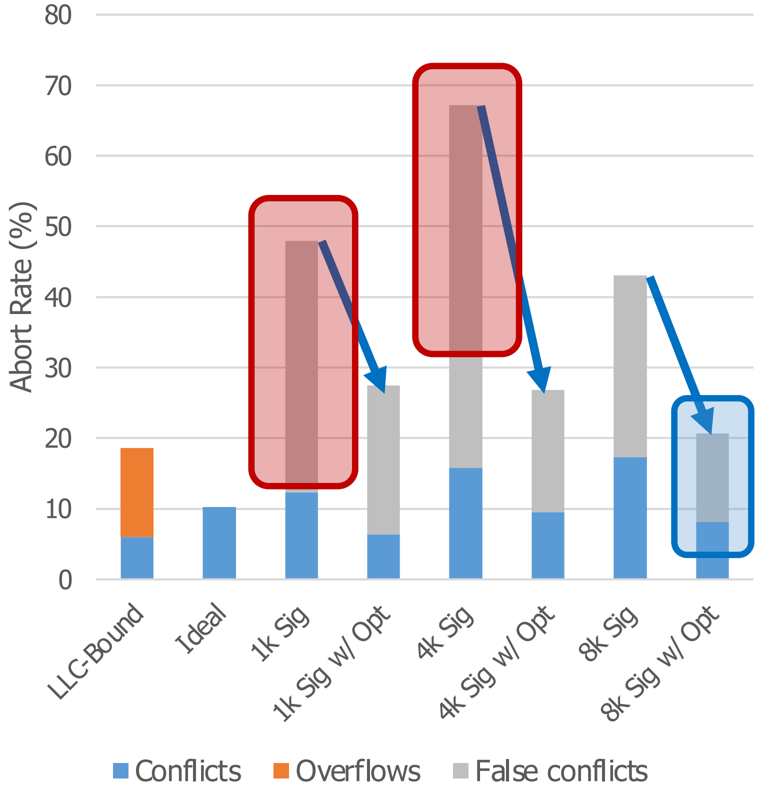
Signature-Isolation renders conflict-detection more accurate

Due to very-high false-positive rates

Evaluation – Hybrid-Index KV-Store

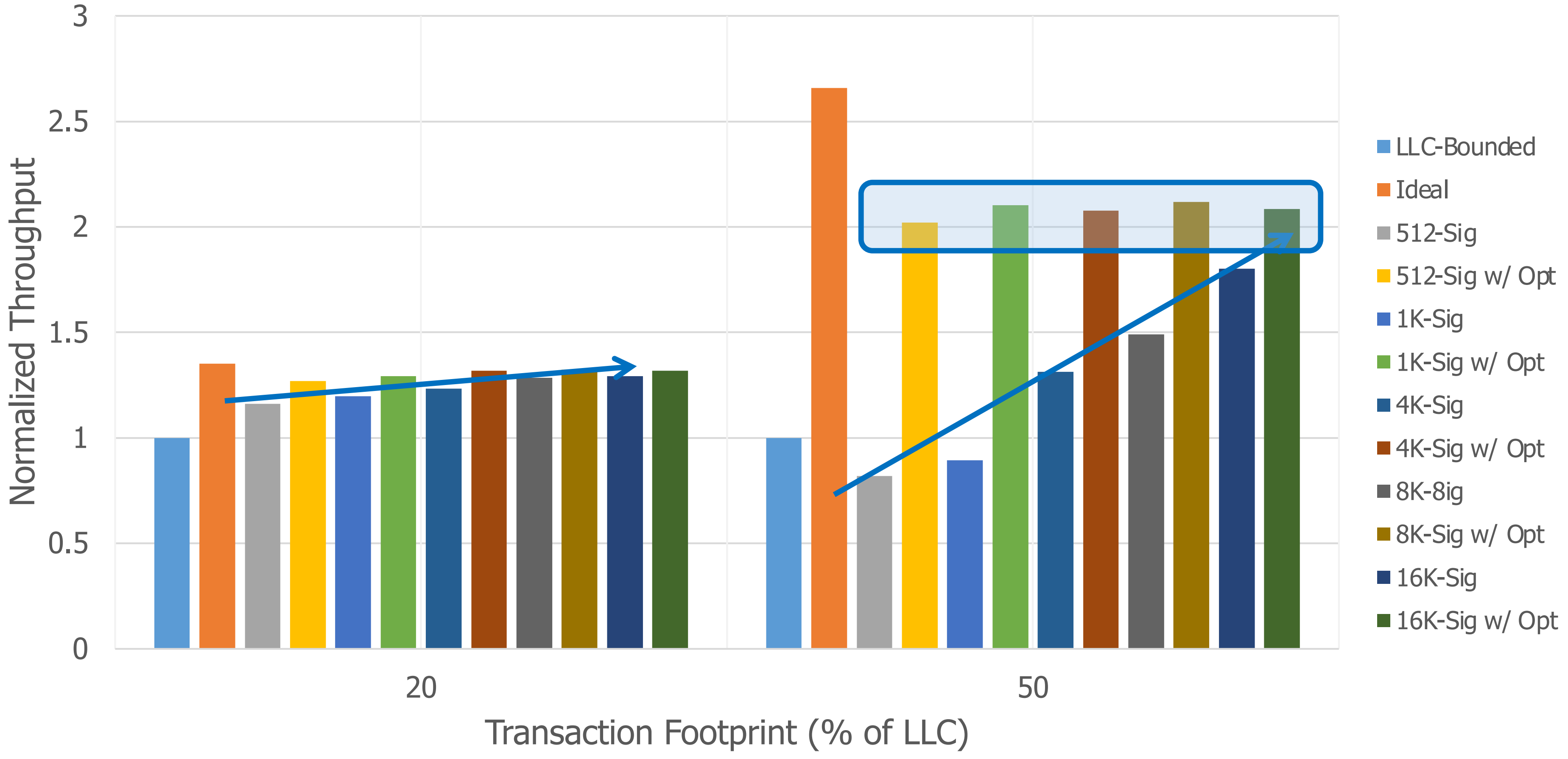
- Small signatures even slower than LLC-Bound
  - Too many false-conflicts
- Signature-isolation greatly improve throughput
  - By reducing false-positive rates





Evaluation – Signature Sizes

- PMDK benchmarks (HashMap, B-Tree, RB-Tree, SkipList)
- Larger signatures ➔ Less false-positives ➔ Higher Throughput
- Signature-isolation effectively reduces false-positives even with small size (e.g., 512-bit)



Conclusion

- UHTM: Unbounded HTM for a DRAM/NVM Memory System

Unlimited Conflict Detection

- Staged conflict detection
  - On-chip cache ➔ Cache-coherence protocol
  - Off-chip memory ➔ Address-Signatures + Isolation Technique

Hybrid DRAM/NVM Support

- Hybrid eager/lazy logging for fast COMMIT
  - DRAM data ➔ Undo-logs (eager)
  - NVM data ➔ Redo-logs (lazy)

Unbounded Hardware Transactional Memory for a Hybrid DRAM/NVM Memory System

Jungi Jeong[§], Jaewan Hong[†], Seungryoul Maeng[†], Changhee Jung[§], and Youngjin Kwon[†]

[§] Purdue University
[†] KAIST

International Symposium on Microarchitecture (MICRO) 2020

Contact Info: jungijeong@purdue.edu

Backup Slides

Two Limitations

2. Limited Transaction Boundary

- Requires programmers to implement the fallback path

```
while (1) {
  int status = tx_begin();
  if (status == XBEGIN_STARTED) {
     /* transaction body:
         fast-path */
      tx_end(); return;
  }
  if (!(status & XABORT_RETRY)) {
     /* user-defined handler
         for overflow:
         slow-path */
  }
  /* User-defined handler
      for non-overflow: retry */
}
```

1. User-defined slow-path
➔ programmer burden

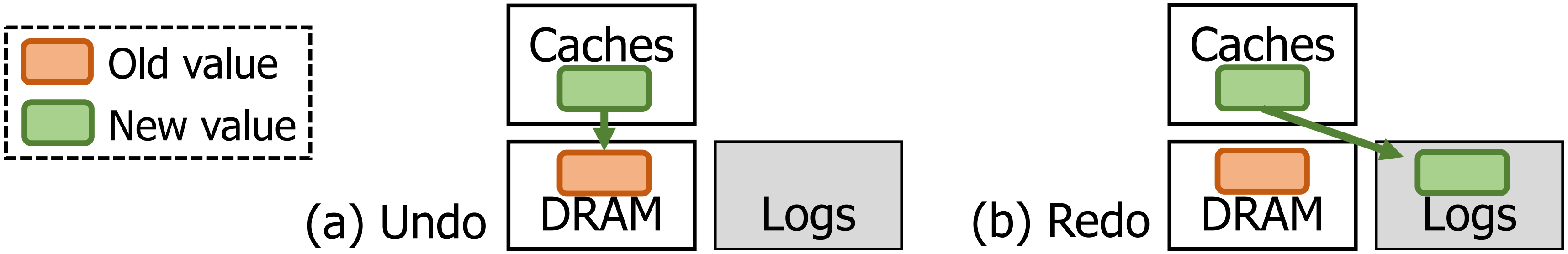
2. Serialized execution
➔ throughput degradation

Hybrid DRAM/NVM Support

DRAM/NVM Hybrid Memory Support

- DRAM only requires atomicity
  - No cache-flush
  - Logging only **overflowed** blocks (HTM handles within caches)

- Undo vs. Redo when LLC-overflow of DRAM data:



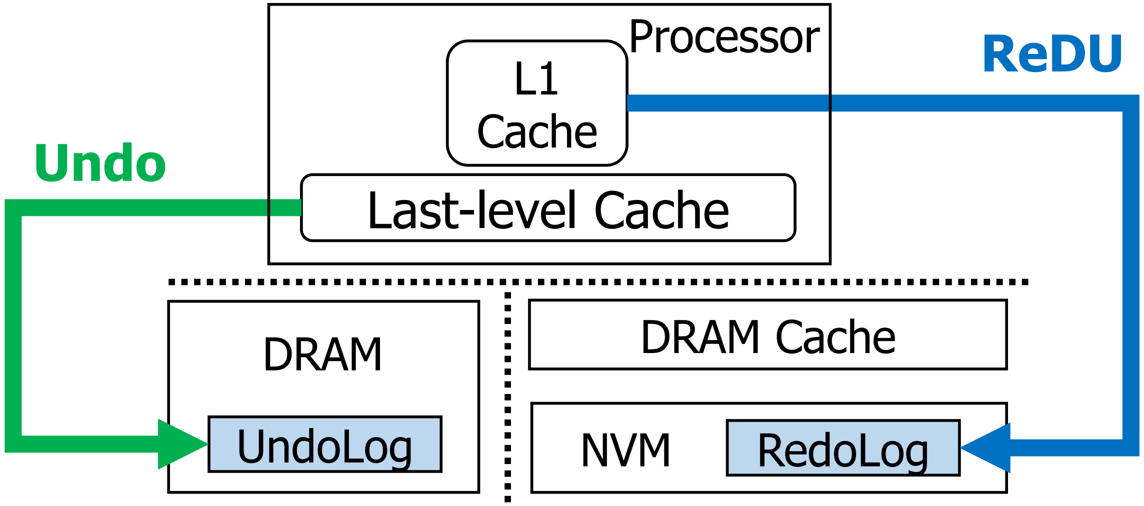
(a) Undo

(b) Redo

Load miss on LLC:	Direct reads	Search logs first
When commit:	Commit immediately	Copy before commit
When abort:	Copy before abort	Abort immediately

Hybrid DRAM/NVM Support

- NVM ➔ ReDU
  - Every store is logged for crash consistency
- DRAM ➔ Undo-log
  - LLC-overflow blocks are logged



Hybrid DRAM/NVM Support

- Diverse data placements & requirements
  - Caches → overwrite & write-back if dirty
  - DRAM → overwrite & undo-logging overflowed blocks only
  - NVM → redo-logging for every NVM store for crash consistency

Common ↔ Less-common ↔ Rare

	Commit	Abort	Power-Failure
On-chip caches	FAST	Invalidate	N/A
DRAM	FAST	Undo-Log (overflow-only)	N/A
NVM	FAST	Redo-Log	Redo-Log